Howto: C-Programmierung

Thursday, 13 October 2022 11:40

① Scopes / Sichtbarkeitsbereiche

Ein Bereich "zwischen" { ... } wird scope (Sichtbarkeitsbereich) genannt.
Variablen, die in einer scope deklariert wurden, sind außerhalb der scope unsichtbar !

```
void f()
{
  int a = 0;
  {
    a++;        ok
  }
  a++;          ok
}
int b = a;      Fehler    Compilation error: 'a' was not declared in this scope
```

scope von a

Scopes sind sehr wichtig, damit es bei größeren Programmen nicht ständig zu Kollisionen mit zufällig identischen Variablennamen kommt.

→ Merke, deklariere eine Variable in der kleinsten (innersten) möglichen scope, um nicht unnötig andere Programmbereiche mit Variablennamen zu verschmutzen !

Variablen, die überhaupt nicht von {, } eingeschlossen sind heißen globale Variablen. Alle anderen Variablen heißen lokale Variablen.

```
int a = 0;      //a ist eine globale Variable
void f()
{
  long b = 11; //b ist eine lokale Variable
  a++;
  b++;
}
```

Werden Variablen mit gleichen Namen in verschachtelten scopes declariert, so ist die Variable äußeren scope zeitweise nicht mehr zugänglich – die Variable ist überdeckt.

```
int a = 0;      //a ist eine gloale Variable
void f()
{
  long a = 11; //lokale Variable a überdeckt globales a in dieser scope
  a++;
  Serial.println(a);  //lokales a ist jetzt 12
}
Serial.println(a);  //globales a ist weiterhin 0
```

kein Zugriff auf globales a möglich

② Funktionen

In der Programmiersprache C sind Funktionen nahezu die einzige Möglichkeit, um das Programm zu strukturieren also übersichtlich und nachvollziehbar für einen Programmierer zu gestalten.

In anderen Programmiersprachen, werden weitere Strukturen unterstützt, um den Code nachvollziehbar und wartbar zu machen.
classes / Klassen, namespaces, templates, generics, decorators, etc...

Jede C-Funktion kann höchstens einen Return-wert (Rückgabewert) haben. Liefert die Funktion nichts zurück muss in der Deklaration der Datentyp void verwendet werden. Auch die return Anweisung kann in diesem Fall verzichtet werden.

```
void f()
{
}

int sepp(float x)
{
  return (int) x;
}

int hias(float x, float y)
{
  return (int) (x+y);
}
```

- Return-Datentyp der Funktion
- Funktionsname (keine Schlüsselworte, Variablen, andere Funktionen) ansonsten beliebig
- Argument/Argumente der Funktion mit Datentyp
- return Anweisung (sofortiger Rücksprung zur aufrufenden Funktion)

Übungen: → Schreiben Sie eine Funktion f, die "nichts macht".
→ Schreiben Sie eine Funktion quadrat, die das Quadrat eines int berechnet
→ Schreiben Sie eine Funktion hoch drei, die mithilfe

```
void setup()
{
  long a;
  double sqrt(double x);
  void notaus(int x)
  {
    if(x==0) exit();
  }
```

```
double a;
void loop()
{
  unsigned char a;
  unsigned char b;

  notaus(a);
}sqrt(4);
```

```
void f()
{
}


int quadrat(int x)
{
  return x*x;
```

→ Schreiben Sie eine Funktion quadrat, die das Quadrat eines int berechnet.

→ Schreiben Sie eine Funktion hochdrei, die mithilfe der Funktion quadrat, x^3 für einen Integer bestimmt.

→ Vertauschen Sie die Reihenfolge der Funktionen quadrat und hochdrei.

→ Schreiben Sie eine Funktion umfang_rechteck, die den Umfang eines ganzzahligen Rechtecks zurückliefert.

```
}

int quadrat(int x)
{
  return x*x;
}

int hochdrei(int x)
{
  int y = quadrat(x);
  return y*x;
}

int umfang_rechteck(int a, int b)
{
  return 2*(a+b);
}
```

Achtung

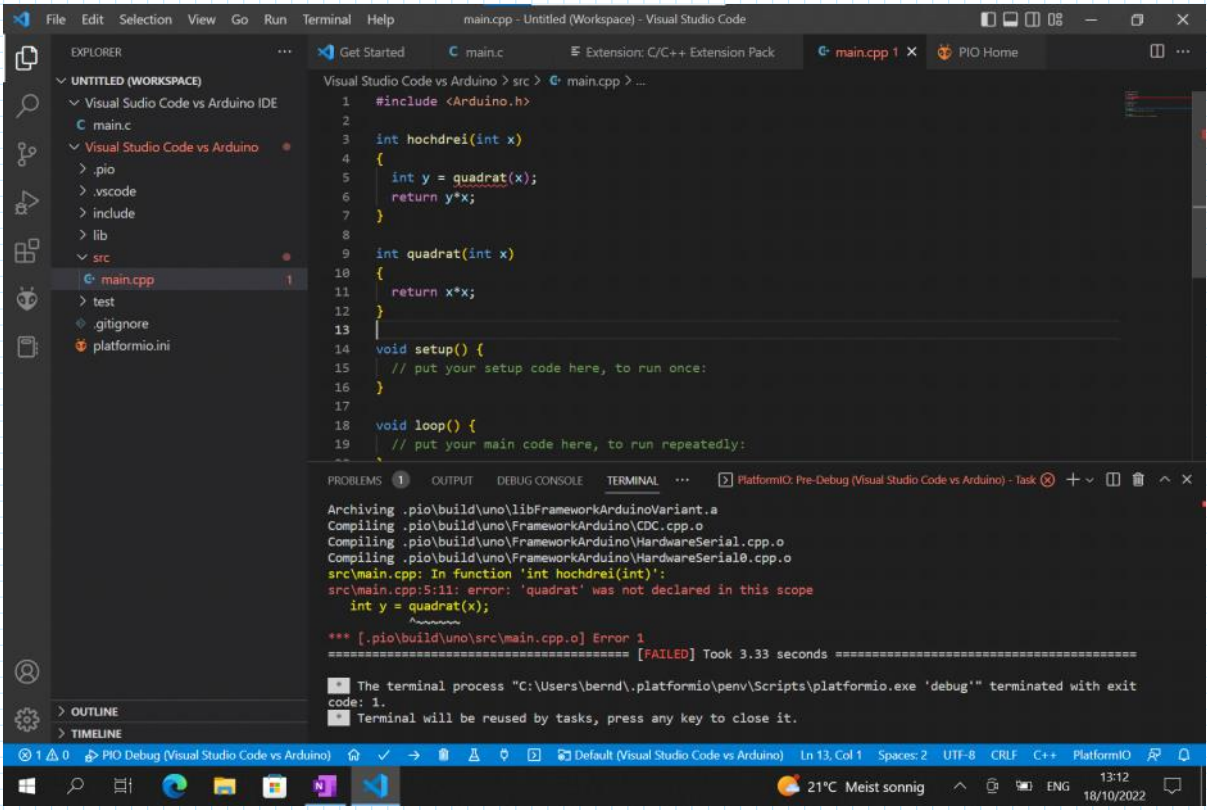
Normalerweise darf man in C und C++ keine Variable und Funktionen verwenden, bevor sie declariert wurden. Die Arduino Entwicklungsumgebung ist in dieser Hinsicht eine Ausnahme, da vorab sogenannte forward-declarations automatisch generiert werden.

Lösung: → Teste folgenden Code auf Visual Studio Code

```
int hochdrei(int x)
{
  int y = quadrat(x);
  return y*x;
}

int quadrat(int x)
{
  return x*x;
}

int main()
{
}
```



Die Lösung zu diesem Problem ist eine sogenannte forward declaration: Funktionskopf mit Strichpunkt.

```
int quadrat(int x); //Dies ist ein forward-declaration

int hochdrei(int x)
{
  int y = quadrat(x);
  return y*x;
}

int quadrat(int x)
{
  return x*x;
}

int main()
{
}
```